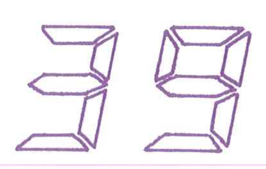



LA REVUE QUI CONSOLE LES BOURSIERS MALCHANCEUX

NOVEMBRE 1987

IBM

WALL STREET TELEGRAMME: SITUATION BOURSIERE DESESPEREE -
CONSEILLONS INVESTISSEMENT
SUR TURBO-FORTH

## EDITORIAL

TURBO-Forth se porte bien, merci. Sa finition est en cours, mais afin de ne pas trop vous faire attendre, nous diffusons déjà une version d'évaluation (version en l'état) permettant de programmer les exemples fournis dans ce numéro. Certains vont hurler: "comment! un nouveau FORTH! pourquoi ne pas garder celui déja existant...". Mais considérons le point de vue du programmeur; voila une personne ayant un travail à réaliser; première étape, définir un squelette et le tester. Si le modèle est satisfaisant, on l'habille. Si le programme est destiné à la commercialisation, il faut particulièrement soigner l'ergonomie. Or, quand le programme devient particulièrement important, comme FORTHLOG II, un rajout trop important impose un remaniement des blocs très délicat. Avec un fichier ASCII et un bon traitement de texte, tout se simplifie. Un FORTH compilant des fichiers ASCII ne s'appelle plus F83, d'où l'idée de nommer ce nouveau produit TURBO-Forth. Finis les blocs, les FCBs, les buffers de 1K. Souhaitons maintenant que vous fassiez bon accueil à ce nouvel arrivant et qu'il vous redonne goût à la pratique d'un langage de programmation très performant.

## SOMMAIRE



Toute reproduction, adaptation, traduction partielle du contenu de ce magazine sous toutes les formes est vivement encouragée, à l'exception de toute reproduction à des fins commerciales. Dans le cas de reproduction par photocopie, il est demandé de ne pas masquer les références inscrites en bas de page, et dans les autres cas de citer l'ASSOCIATION JEDI (association loi 1901).

Nos coordonnées: ASSOCIATION JEDI 17, rue de la Lancette 75012 PARIS
tel président: (1) 43.40.96.53
tel secrétaire: (1) 46.56.33.67

# GESTION DE VARIABLES LOCALES

F83 Laxen et Perry - MSDOS - CP/M
Turbo-FORTH - MSDOS

par Marc PETREMANN

A chaque article suffit sa peine. La dernière fois que nous parlions de variables locales, on avait introduit le concept TO (JEDI n°25,26 et 30). Or, bien qu'intéressant, ce principe ne s'affranchit pas de la génération d'en-têtes dans le dictionnaire, en-têtes parfois génants si on est un peu puriste et que l'on souhaite travailler avec des variables locales "PASCAL-like".

Le programme que je propose ci-après n'est pas de mon crû. Il a déja été diffusé dans Dr DOBBS JOURNAL. Je n'ai fait que l'adapter au F83 de Laxen et Perry, le FORTH le plus répandu.

La partie la plus difficile à mettre au point a été la suppression des en-têtes de variables après utilisation.

Le principe de ces nouvelles variables locales fait appel à deux nouveaux mots de définition, <DECLARE qui débute la déclaration des variables, <DEFINE qui commence la définition d'un mot de manière identique au mot : (deux-points) et le mot >> qui ferme la définition. Entre <DECLARE et <DEFINE, les variables sont crées par VAR et LOCAL. Exemple:

```
<DECLARE   VAR N1   VAR N2   LOCAL SOMME
<DEFINE    ADDITION
           N1 N2 + SOMME LET   ( somme:=n1+n2)  >>
```

A l'exécution, les données empilées sont affectées de la manière suivante aux variables déclarées par VAR:

```
n1 n2 ADDITION
|  |
|  └── n1 passe dans N1
└───── n2 passe dans N2
```

La définition des variables locales par LOCAL permet d'exploiter des variables dont les données sont initialisées à zéro. En sortie de procédure, il faut laisser sur la pile le résultat des opérations. Exemple:

(n1+n2)*n3=n4   n1+n2/2=n5   n4+n5=> valeur à sortir

```
<DECLARE   VAR N1   VAR N2   VAR N3
           LOCAL N4 LOCAL N5
<DEFINE    OPERATION  ( n1 n2 n3 --- n)
           N1 N2 + N3 *  N4 LET
           N2 2/ N1 +    N5 LET
           N4 N5 +   >>
```

Ca surprend, mais c'est très efficace. Et pour vous en convaincre, tapez SEE OPERATION, vous comprendrez mieux comment travaillent les variables locales.

Listing 1:

```
HEX
VOCABULARY ALEPH
ONLY FORTH ALSO ALEPH ALSO
ALEPH DEFINITIONS
CREATE #LST 50 ALLOT
    #LST 4E +         \ valeur du point de débordement
CONSTANT #LS0         \ val initiale du pointeur de pile
VARIABLE #PTR         \ pointeur de pile lambda
VARIABLE DP-SAVE      \ sauvegarde de DP
#LS0 #PTR !
: >L
  #PTR DUP @ DUP 2-  DUP #LST  >
  IF   ROT ! !
  ELSE TRUE ABORT" Pile lambda pleine"
  THEN ;
: L>
  #PTR  DUP @ 2+ DUP ROT ! @ ;
: LS@
  #PTR @ + @ ;
: LS!
  #PTR @ + ! ;
: #POP  #PTR +! ;

CREATE #LAMBDA 100 ALLOT
DECIMAL
  4 CONSTANT 4     6 CONSTANT 6     8 CONSTANT 8
 10 CONSTANT 10   12 CONSTANT 12   14 CONSTANT 14
: [2-14]   ( n --- )
  CASE  2 OF    ['] 2 ,     ENDOF
        4 OF    ['] 4 ,     ENDOF
        6 OF    ['] 6 ,     ENDOF
        8 OF    ['] 8 ,     ENDOF
       10 OF    ['] 10 ,    ENDOF
       12 OF    ['] 12 ,    ENDOF
       14 OF    ['] 14 ,    ENDOF
       DUP [COMPILE] LITERAL
  ENDCASE ;
HEX
CREATE LINK-ARRAY   8 ALLOT
FORTH DEFINITIONS
: <DECLARE               \ Définit opérateur LAMBDA
  #LS0 #PTR !            \ RESET pointeur pile   LAMBDA
  CURRENT @ DUP CONTEXT !  \ restaure contexte
  LINK-ARRAY 8 CMOVE     \ sauve les liens voc   courant
HERE DP-SAVE !          \ sauve pointeur de dictionnaire
#LAMBDA DP !             \ déplace pointeur dictionnaire
0 >L   ;                      \ met zéro en haut de pile

ALEPH DEFINITIONS
\     Utilisé dans une définition ALEPH, sous la forme:
\     <DECLARE    VAR <nom> ... LOCAL <nom> ...
\     <DEFINE <nom> ...définition...            >>

FORTH DEFINITIONS
: LET
  ['] LS! HERE 2- ! ; IMMEDIATE
: VAR  ( --- <nom> )
  CREATE   L> 1+ DUP >L 2* C, IMMEDIATE
  DOES>    C@ [2-14]  COMPILE LS@ ;
: LOCAL ( --- <nom> )
  CREATE   L> 1+ 100 +
           DUP >L OFF AND 2* C, IMMEDIATE
  DOES>    C@ [2-14]  COMPILE LS@ ;
HEX
: <DEFINE
  DP-SAVE @    DP !
  !CSP CURRENT @ CONTEXT !
  CREATE   HIDE  2 LS@ 100 /  ?DUP
  IF
     BEGIN
        COMPILE 0   1- ?DUP 0=
     UNTIL
  THEN
  2 LS@  OFF AND  ?DUP
  IF
     BEGIN
        COMPILE >L  1- ?DUP 0=
     UNTIL
  THEN
  ]  ['] : @  LAST @ NAME> !   ;
: >>
  2 LS@  OFF AND  2* [2-14]
   COMPILE #POP  LINK-ARRAY CURRENT @ 8 CMOVE
   LAST @ CURRENT @ HASH  DUP @ LAST @ 2- !
   LAST @ 2- SWAP !  [COMPILE] ;  ; IMMEDIATE
ONLY FORTH ALSO
DECIMAL
```

Nota: le présent listing est exempt de toute erreur, il a été repris directement sur le disque après traitement par Turbo-FORTH. Si vous le retapez dans des blocs pour compilation par F83, il devrait fonctionner sans accroc.

Normalement, il est opérationnel sur les versions F83 sous MSDOS (CompIBM) et F83 CP/M (AMSTRAD toutes versions).

Les améliorations possibles:
- optimisation en langage machine le LS@ et LS!.
- déplacer la zone #LST en dehors du dictionnaire et gérer un mécanisme de pile descendante.
- intégrer le mécanisme de gestion de variables locales à la définition de : et ; avant méta-compilation. Le mot <DECLARE serait conservé. S'il n'est pas utilisé avant une définition deux-points, aucune différence ne devrait apparaître entre un mot deux points ordinaire et votre définition créée par votre FORTH nouvelle version. Dans le cas contraire, le mot L> devrait apparaître à la décompilation.

A titre d'exemple vraiment concret, voici une application des variables locales dans l'article qui suit.

## FONCTIONS GRAPHIQUES EN FORTH

F83 Laxen et Perry - MSDOS
Turbo-FORTH - MSDOS

par Marc PETREMANN

Le premier mot à définir est le mot MODE. Celui-ci a été testé avec succès sur le PERSONNA 1600 de LOGABAX et fonctionne plus ou moins bien sur d'autres systèmes en fonctions des modes graphiques gérés par votre système compatible.

Le mot DARK primitivement défini par Laxen et Perry nettoie l'écran par sélection systématique du mode graphique 2 (mode texte 25x80 couleur). Il faut donc définir le mot MODE de manière à le faire réexécuter systématiquement par DARK sans revenir au mode 2. C'est ce que fait (MODE) en reprenant le contenu de la variable GRMODE (contraction de GRaphic-MODE). Le mode graphique doit être choisi entre 0 et 7. Pour exemple, le mode 6 bascule votre affichage en mode graphique 640x200 (si carte controleur écran adéquate).

```
VARIABLE GRMODE  2 GRMODE !
CODE (MODE)  ( ---   n entre 0 et 7)
  GRMODE #) AX MOV   16 INT   NEXT  C;
' (MODE)  IS DARK
: MODE ( n ---)
  GRMODE !  DARK  ;
```

Ensuite, DARK est revectorisé sur (MODE) et MODE stocke le mode graphique demandé dans GRMODE avant exécution de DARK. Si vous trouvez plus simple...

```
\ fonctions graphiques
VARIABLE (GCOLOR)
CODE GCOLOR ( n ---)
  (GCOLOR) # BX MOV   0 [BX] POP   NEXT END-CODE

VARIABLE XPLOT          VARIABLE YPLOT
HEX
CODE PSET  ( x y ---)
  DX POP   YPLOT # BX MOV   DX 0 [BX] MOV   ( YPLOT:=dx)
  CX POP   XPLOT # BX MOV   CX 0 [BX] MOV   ( XPLOT:=cx)
  0C # AH MOV
  (GCOLOR) #) AL MOV   10 INT   NEXT  END-CODE
DECIMAL
```

Le mot GCOLOR sélectionne la couleur du tracé en mode graphique. Exemple:

```
1 GCOLOR
```

Le mot GCOLOR définit en code machine peut aussi se définir en FORTH sous la forme:

```
: GCOLOR   (GCOLOR) ! ;
```

Les variables XPLOT et YPLOT sont chargées de mémoriser les coordonnées du point tracé par PSET. Exemple:

```
100 120 PSET   XPLOT ?  YPLOT ?
affiche 100 120
```

Voici le gros morceau, la définition de LINE. Si IBM (et ses clones) ont jugé utile de disposer d'une fonction pré-programmée pour faire des points en mode graphique, il n'en est pas de même pour tracer une ligne. Alors pour parer au plus pressé, et à titre d'exemple, illustrons ce qui a été écrit dans la précédente rubrique (les variables locales):

```
<DECLARE   VAR XD        VAR YD
           VAR XF        VAR YF
           LOCAL DX      LOCAL DY
           LOCAL XINCR   LOCAL YINCR
           LOCAL CUMUL   LOCAL X
           LOCAL Y
<DEFINE LINE
   XD X LET   YD Y LET    XD XF <
   IF  1  ELSE  -1  THEN
   XINCR LET
   YD YF <
   IF  1  ELSE  -1  THEN
   YINCR LET
   XD XF - ABS DX LET
   YD YF - ABS DY LET    DX DY >
   IF DX 2/ CUMUL LET     DX 1+ 1
     DO X XINCR + X LET
        CUMUL DY + CUMUL LET   CUMUL DX >=
        IF CUMUL DX - CUMUL LET
           Y YINCR + Y LET
        THEN
        X Y PSET
     LOOP
   ELSE
   DY 2/ CUMUL LET   DY 1+ 1
     DO Y YINCR + Y LET
      CUMUL DX + CUMUL LET     CUMUL DY >=
      IF CUMUL DY - CUMUL LET
         X XINCR + X LET
      THEN
      X Y PSET
    LOOP
  THEN  >>
```

et s'utilise en précisant les coordonnées de départ et d'arrivée.

```
: LINETO ( xf yf ---)
 >R >R   XPLOT @   YPLOT @   R> R>  LINE  ;
```

Le mot LINETO trace un segment en reprenant comme point de départ les coordonnées du dernier point tracé.

```
<DECLARE  VAR XD   VAR YD
          VAR XF   VAR YF
<DEFINE  BOX
   XD YD XF YD LINE
   XF YF LINETO
   XD YF LINETO
   XD YD LINETO  >>
```

Le mot BOX trace un rectangle. Inutile de vous faire un dessin si vous essayez ce mot.

```
: BOXTO ( xf yf ---)
 >R >R  XPLOT @   YPLOT @  R> R>  BOX  ;
```

De même, BOXTO se passe de commentaire. On applique à BOX ce que LINETO fait subir à LINE.

```
<DECLARE  VAR XD    VAR YD
          VAR XF    VAR YF
<DEFINE   BOXF
   XF 1+ XD
   DO I YD 1- I YF LINE
   LOOP  >>
```

Et pour achever notre délire, BOXF dessine un rectangle plein.

Les améliorations possibles:

- réécrire LINE en langage machine (c'est un gros travail, mais en décompilant LINE, vous serez aidé).

- sécuriser LINE et PSET afin de ne pas tracer en dehors de l'écran, ceci en fonction du mode graphique sélectionné. Si on est en mode non-graphique, signaler par un message d'erreur.

- on peut connaître par MSDOS le type de contrôleur écran. Définir une routine autorisant l'accès aux différents modes si le contrôleur adéquat est disponible.

Nota: concernant Turbo-FORTH, la définition de MODE et la revectorisation de DARK sur (MODE) est implantée en standard.

Voilà, assez travaillé du chapeau, je vais me coucher, bonne nuit.

## GESTION DES ATTRIBUTS MINITEL

79-STANDARD
par J.C. LEMAIGRE

Objet: gestion des attributs de l'écran: clignotement, couleur, curseur, etc...

Chaque attribut demande l'envoi d'une séquence de caractères commençant généralement par ESC suivi de 1 ou 2 caractères de définition. Pour exemple, le pasage en double longueur des caractères sur un fond gris avec clignotement devra s'écrire:

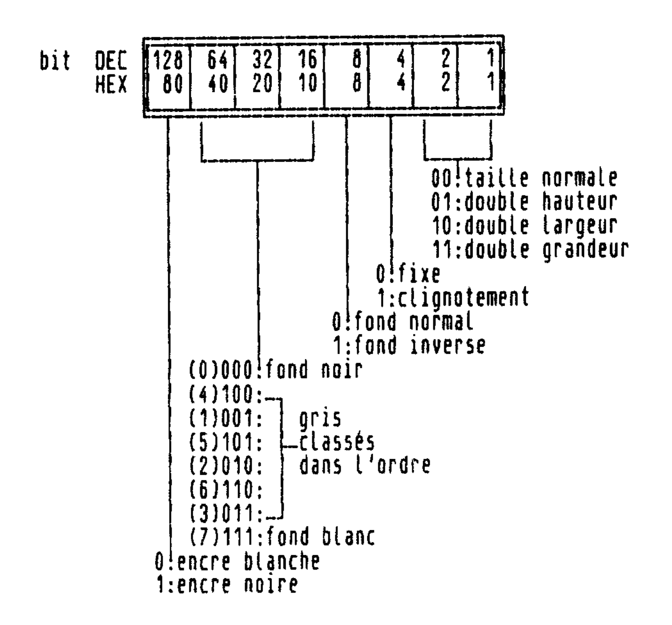
```
HEX : ESC  1B EMIT ;
ESC 4E EMIT  ESC 42 EMIT  ESC 48 EMIT
DECIMAL
```

En cas de mise en page d'un texte, ceci n'est pas sans alourdir le texte source, d'autant plus que toute modification directe de la position du curseur siginifie la fin des attributs précédents.

Les séquences principalement utilisées sont souvent assez réduites, on charge le fond, l'encre, la taille des caractères. Il est tentant de compresser le codage des attributs pour éclaircir le texte source.

ENCODAGE DES ATTRIBUTS

1) Définition du code sur 1 octet.




Définition de la position du curseur:

```
: POSITION ( n°ligne n°colonne ---    en décimal)
  64 + SWAP 64 + 31 EMIT EMIT ;
HEX
: ATRIB  ( n1 ---    attributs minitel)
DUP 03 AND 4C + ESC EMIT
DUP 04 AND
IF     ESC 48 EMIT
ELSE   ESC 49 EMIT   THEN
DUP 08 AND
IF     ESC 5D EMIT   THEN
DUP 70 AND 10 /
ESC 50 + EMIT
80 AND
IF     ESC 40 EMIT
ELSE   ESC 47 EMIT   THEN  ;  DECIMAL
```

Utilisation:

```
xx ATRIB
```

Note: les attributs commencent après un espace pour la couleur du fond qui n'est valide que sur la ligne en cours, tous les caractères sont valides sur la page, sans espace préliminaire. Il faut définir la position avant les attributs. Pour les faibles en calcul mental, l'utilisation d'une calculatrice de poche est toléré (Ndlr: et si on a FORTH sous la main, peut-on travailler en binaire?... Rdlr-réponse de la rédaction-: bon sang, mais c'est bien sûr!...)

## LIBERATEUR DE BLOC ECRAN

79-STANDARD
par J.C. LEMAIGRE

Lorsqu'on dispose d'un vieux Forth 79-STANDARD enROMé, quelques petits utilitaires de gestion facilitent bien la vie, surtout si l'on teint à ce que ses programmes aient une tête bien ordonnée pour une maintenance future.

C'est le cas pour ces deux petites choses qui travaillent régulièrement pour moi (quand on a pas de tête, il faut avoir des esclaves...). La première s'appelle LIBERE. Son rôle est de libérer l'écran que l'on désire utiliser, en poussant le contenu de celui-ci vers le suivant si celui-ci est libre, sinon... vous me suivez?

```
( editeur #DECAL E-ECRANS LIBERE)
: #VIDE  BLOCK SWAP -TRAILING  SWAP DROP ;
: RECHERCHE-VIDE
BEGIN  1+ DUP 1024 SWAP
       #VIDE 0=
UNTIL  ;
: #DECALE-ECRANS   ( n1 n2 --- décale les écrans n1 à n2
d'un écran)
                 ( suppose que n2+1 est libre)
SWAP DUP ROT 1+ DO I 1- I COPIE  1- +LOOP NEW FLUSH  ;
: LIBERE  ( n1 ---)
DUP RECHERCHE-VIDE 1-
#DECALE-ECRANS  ;
```

Le deuxième de ces esclaves est complémentaire de celui-ci, car après l'utilisation de plusieurs LIBERE et de quelques SWAP-ECRANS pour organiser son programme, on a une forte tendance à s'exclamer "mais M... (Ndlr: et non M... comme pourrait le croire Mr BOUYGUES) où est donc ce mot TOTO (re-Ndlr: on propose le mot TOTO comme réservé par le système...)". La seule chose dont on soit encore à peu près sûr est l'écran où commence ce ******** (censuré (...ou bétonné, comme dirait M.POLAC...)) de programme. C'est le rôle de FIND qui recherche dans tous les écrans à la suite du n° spécifié le mot TOTO. Utilisation:

```
20 FINO TOTO
```

Ndlr: en F83, sous éditeur, cette fonction existe, elle s'appelle S et s'utilise en tapant n S texte et recherche

le texte à partir du bloc courant jusqu'au bloc numéro n.

```
( FIND TEXTE)
: #FD1  ( ---)
        ( troouve mot suivant et --> PAD)
32 WORD HERE C@ 1+ CMOVE
0 HERE C! ;
: #CMP  ( COMPAR 2 CHAINE AD1 AD2 LG)
0 0 ROT SWAP DO DROP DUP C@ ROT DUP C@
ROT = 0= DUP IF LEAVE THEN
ROT 1+ ROT 1+ SWAP ROT LOOP
ROT DROP SWAP DROP 0= ; ( si flag 1)
0 VARIABLE #TROUVE
: #FD2  ( n° BLK--0/add dans TROUVE)
0 #TROUVE !
BLOCK DUP 1024 + SWAP DO
  I 32 ENCLOSE DUP NOT IF DROP 104 THEN
  >R SWAP - PAD C@ =
  IF DUP PAD C@ PAD 1+ SWAP #CMP
     IF DUP #TROUVE !  THEN
  THEN R> + 1- R> DROP >R
  #TROUVE @ IF LEAVE THEN
LOOP ;
( FIND TEXTE)
: FIND  ( recherche un mot à partir )
        ( du n° de block spécifié)
#FD1 ( transfre MOT dans PAD)
CR ( CR ." à partir du BLOCK N°: " #IN )
0 #TROUVE !
BEGIN DUP #TROUVE @ NOT ?TERMINAL
      NOT AND
WHILE 13 EMIT DUP . #FD2 1+
      2SIDES NOT
      IF ." NON TROUVE" ABORT  THEN
REPEAT
#TROUVE @
IF 1- LIST
ELSE CR ." INTERROMPU EN " 1- .
THEN ;
```

Le mot SIDES ( n1 --- n1 flag) renvoie 0 en flag si le n° d'écran n'est pas dans l'espace disque.

## PROBLEME EXISTENTIEL

79-Standard
par J.C.LEMAIGRE

Je travaille énormément en "overlay" sur mon éditeur, car mon espace mémoire est de 8k, éditeur compris. Je vois les PCistes sourire, mais essayez donc de faire tourner ADA là-dedans!. Bref, si je me rappelle à peu près où sont logés tous les petits bouts nécessaires, il est difficile de se rappeler si le dernier mot utilisé a détruit ou pas le NCASE ou le .S un peu spécial nécessaire. Alors ce petit truc est très pratique en restaurant l'environnement adéquat.

```
: EXISTE?  IF 2DROP DROP ELSE LOAD THEN ;

129 -FIND NCASE EXISTE?
 48 -FIND .S    EXISTE?
```

Je sais, c'est un peu court pour remplir un journal.

## DECOMPRESSION DES FICHIERS AU LANCEMENT DE F83

F83 Laxen et Perry MSDOS - CP/M
par Marc PETREMANN

Suite à une question posée par téléphone par Mr PERRET de FLERS, et qui est la suivante: " comment démarrer FORTH sans passer par la décompression de fichiers au lancement de RUNME.COM", j'apporte les précisions suivantes.

Quand vous recevez la disquette contenant F83, le programme à démarrer s'appelle RUNME.COM. Ce programme vous invite à expanser les fichiers disponibles sur le disque. Si vous ne voulez pas expanser les fichiers, tapez sous MSDOS:

```
EMPTY
' HELLO IS BOOT
SAVE-SYSTEM F83.COM
```

La procédure normale d'expansion de fichier impose la présence de deux lecteurs de disquette, respectivement A: et B:. La disquette origine est introduite en A:, les disquettes destination dans B:. Prévoir deux disquettes formatées.

Pour utiliser F83, il n'est pas nécessaire de décompresser les fichiers marqués par l'extension.HUF. Ces fichiers contiennent le méta-compilateur F83 et les fichiers source ayant servi à générer F83. Eh oui, F83 est écrit et généré par F83. Mais si vous disposez d'un disque dur ou si vous ne disposez que d'un seul lecteur de disquette voici comment opérer une décompression "manuelle":

1) sur disque dur sous MSDOS, créer un sous-répertoire FORTH en tapant:

```
MKDIR FORTH
```

copier la disquette contenant F83 sur le disque dur:

```
COPY A:*.* C:\FORTH
```

passer sur le disque dur:

```
C:
CD\FORTTH
```

vérifiez en tapant DIR et lancer RUNME.COM. Opérer comme décrit précédemment pour générer F83.COM.

Repasser sous DOS et relancer RUNME. On décompresse un fichier en tapant:

```
EXPAND fichier.HUF fichier.BLK
```

Exemple:

```
EXPAND K86.HUF KERNEL86.BLK
```

Le nom de fichier peut être précédé du n° de drive source ou destination:

```
EXPAND A:K86.HUF C:KERNEL86.BLK
```

Si vous ne disposez que d'un seul lecteur de disquette sous MSDOS:

copier RUNME.COM sur une disquette vierge et formatée; copier un fichier d'extension .HUF sur cette même disquette; lancer RUNME et taper sous FORTH:

```
EXPAND fichier.HUF fichier.BLK
```

L'opération de compression d'un fichier ascii quelconque est activée par COMPRESS fich-orig fich-dest. Exemple:

```
COMPRESS C:TEST.DOC  A:TEST.HUF
```

Sous CP/M, c'est à dire depuis un AMSTRAD, la décompression des fichiers source est activée sous CP/M en utilisant le programme USQ.COM. Exemple:

- mettre USQ sur un disque vierge et formaté;
- copier un fichier d'extension BQK ou HQX;
- taper:

```
USQ EXTEND.BQK
```

Le système vous informe qu'il lance la décompression:

```
EXTEND.BQK --> EXTEND.BLK
```

Si le fichier n'est pas un fichier compressé, l'opération de décompression echoue.

ATTENTION: sur AMSTRAD, la capacité disque risque d'être insuffisante si on décompresse un fichier da grande taille. Pour exemple, le seul fichier META80.BLK fait 240k de capacité disque. Cet avertisement est valable même si on décompresse à l'aide de deux lecteurs disque.

Ce que contiennent les fichiers sous CP/M:

META80.BLK le méta-compilateur et le source du noyau FORTH
EXTEND80.BLK les extensions du système F83
CPU8080.BLK l'assembleur FORTH 8080
UTILITY.BLK les utilitaires F83
KERNEL.HEX le noyau au format HEX. Pour le transformer en .COM, charger l'utilitaire LOAD livré avec la disquette système CP/M:

```
LOAD KERNEL.HEX   ce qui génère KERNEL.COM
```

Le fichier KERNEL.COM est identique à ce qui résulterait d'une opération de méta-compilation. Ce fichier contient un F83 minimal. La poursuite de la méta-compilation est exécutée en tapant:

```
KERNEL EXTEND80.BLK
OK  ( ceci doit être tapé, ce n'est pas un 'prompt')
```

En fin de méta-compilation, taper BYE, prendre note du nombre de pages affiché et taper sous CP/M:

```
SAVE n F83.COM
```

où n est le nombre précédemment noté. Etant entendu que les fichiers EXTEND, UTILITY et CPU contiennent une partie du source FORTH, si le contenu de ces fichiers est modifié, votre nouvelle version de F83 prendra en compte ces modifications.

Contenu utile sous MSDOS:

```
META86.BLK    |
KERNEL86.BLK  |  similaire à META80.BLK de CP/M
EXTEND86.BLK  |
UTILITY.BLK
CPU8086.BLK
HUFFMAN.BLK
CLOCK.BLK
```

Le méta-compilateur est lancé en tapant:

```
F83 META86.BLK
puis sous FORTH
OK  ( ceci doit être tapé)
```

...attendre... FORTH génère le fichier KERNEL.COM

Revenir sous MSDOS et taper:

```
KERNEL EXTEND86.BLK
OK  ( ceci doit être tapé)
```

Attention, si vous méta-compilez sur disque dur, éditez au préalable les blocs 1 des fichiers META86 et EXTEND86. En effet, les fichiers appelés depuis le méta-compilateur sont forcés depuis le lecteur A:. Remplacer A: par C: ou mieux, par rien.

Exemple:

```
remplacer dans META86.BLK, écran 1
FROM A:KERNEL86.BLK 1 LOAD
par
FROM KERNEL86.BLK 1 LOAD
```

Procédez de même dans le fichier EXTEND aux endroits où le contenu de CPU et UTILITY est appelé par FROM.

Voilà, vous savez maintenant comment procéder pour méta-générer Turbo-FORTH.

Note: la sélection du disque dur sous MSDOS peut être activée par le mot C: dont la définition est:

```
: C: [ DOS ] 2 SELECT [ FORTH ] ;
```

## RECUPERATION D'EVALUATION

Le LISP
par J.M. PERRET

Je profite de l'occasion pour vous envoyer ma modeste contribution: cette fonction Le Lisp permet de récupérer la dernière évaluation et de la lier au symbole "à":

```
(de recup ()
(setq à #:toplevel:eval))
```

Un exemple:

```
? (+ 2 2)
= 4
(recup)
= 4

? (+ 1 à)
= 5
```

C'est tout! J'espère faire mieux la prochaine fois.

COURRIER: Cher secrétaire,

Je viens de recevoir (enfin, il y a déjà quelques jours) le n° 38 de JEDI, je suis heureux de voir que JEDI continue. Suite à ma lettre, je vous fait parvenir un "article" (si on peut appeler ça ainsi) (Ndlr: voir ci-après: COMPLEMENT A L'ARTICLE). Je n'ai pas de traitement de texte sur PC. Je l'ai donc écrit avec l'éditeur de Turbo-PASCAL. Mais au fait, d'où me vient cette idée qu'il faut envoyer ses articles sur disquettes PC? Il me semble plutôt que JEDI est imprimé avec un MAC?

L'annonce de Turbo-FORTH est vraiment allèchante et je suis intéressé au plus haut point. Est-il déjà disponible? Si oui, combien coûte-t-il? De la doc, vite, vite!

Je ne connais pas le Forth 83 de Laxen et Perry. J'ai des FORTH qui proviennent de OUF, mais sans doc et sans possibilités graphiques. Où peut-on se procurer ce Forth? Son prix?

Bonne continuation.

René BARONE
13397 MARSEILLE CEDEX 13

REPONSE: Cher adhérent forthien,

Quand vous recevez JEDI, une ou plusieurs feuilles jointes donnent des informations complémentaires, notamment la LETTRE DU SECRETAIRE qui regroupe des infos concernant les produits, les annonces, les offres de services et les directives pour envoyer les articles:

- F83 est disponible au prix de 100 Fr, le manuel au prix de 120 Fr, le tout port compris.

A priori, la solution des articles sur disquettes PC est la plus pratique, car elle permet un remaniement éventuel et une mise en forme aisée de votre prose. Mais un bon article déjà "monté" ne nous rebute pas, à condition que la frappe soit bien contrastée et que la typographie respecte les règles élémentaires. J'ai moi-même donné l'idée de l'éditeur Turbo-PASCAL pour écrire les articles si on ne dispose pas d'un véritable traitement de texte.

suite page 8

# LOGIQUE BINAIRE EN TURBO-PROLOG

Ecrire un programme de logique en utilisant la **PRO**grammation **LOG**ique, c'était logique non !

Ce programme permet d'une part d'évaluer une expression logique en donnant la table de vérité, d'autre part de simplifier une expression logique et aussi de la transformer avec des NAND à 2 entrées.

Les domaines sur lesquels il travaille sont assez restreints. Tout d'abord la définition d'une expression logique (*expr*) faite de manière récursive (c'est aussi cela la puissance de TURBO-PROLOG) puis la définition d'une liste de chaînes de caractères (***liste***).

## 1 . Evaluation d'une expression logique

Les différentes clauses sont définies par le prédicat ***évalue*** chargé d'évaluer en 0 ou 1 l'expression donnée. Ils ne posent pas de problèmes particuliers étant donné les primitives ***bitor***, ***bitand*** et ***bitnot*** présentes en TURBO-PROLOG.

A noter cependant la relation ***X=Z+2*** située après un *bitnot*.

En effet, comme dans presque tous les langages, ***bitnot(0,Z)*** donne Z=-1 <u>au lieu de 1</u> et, ***bitnot(1,Z)*** donne Z=-2 <u>au lieu de 0</u>. Il faut donc ajouter 2 dans tous les cas pour retrouver le résultat "normal". (Cela ne vous rappelle-t-il pas la notion de complémentation à 2 ?).

Une fois le programme lancé avec l'option **Run** nous pouvons évaluer des expressions logiques. La seule contrainte est d'entrer l'expression sous forme d'arbre.

<u>Exemples</u> :

a) *Goal :* ***évalue(ou(booléen(A),booléen(B)),S)*** donne la table de vérité de l'expression **S = A + B** (voir <u>encadré 1</u>).

b) *Goal :* ***évalue(et(booléen(A),booléen(B)),S)*** donne la table de vérité de l'expression **S = A.B** (voir <u>encadré 2</u>).

c) *Goal :* ***évalue(ou(booléen(A),et(booléen(B),booléen(C))),S)*** sort la table de vérité de l'expression **S = A + B.C** (voir <u>encadré 3</u>).

d) *Goal :* ***évalue(ou(booléen(A),et(non(booléen(A)),booléen(B))),S)*** sort la table de vérité de l'expression :

$$S = A + \bar{A}.B$$ (voir <u>encadré 4</u>).

A noter que ce dernier exemple donne les mêmes résultats que le premier.

En effet $A + \bar{A}.B$ peut se simplifier en A + B, règle de simplification qui sera d'ailleurs utilisée par la suite.

## 2 . Simplification d'une expression logique

L'expression logique étant rentrée ici de façon plus conviviale que précédemment (voir exemples ci-après) il était nécessaire de fabriquer un analyseur lexical. C'est le rôle des clauses ***expr_liste*** qui sont chargées de transformer sous forme de liste une expression entrée.

Ainsi l'expression **a+b** sera transformée en la liste ["a","+","b"] alors que l'expression (-a).b+a.(-b) se verra transformée en la liste ["(","-","a",")",".","b","+","a",".","(","-","b",")"].

<u>Quelques remarques à propos de la notation</u> :

L'expression (-a).b+a.(-b) sera entrée pour $\bar{a}.b + a.\bar{b}$. En effet, la fonction **ET** étant prioritaire sur la fonction **NON**, l'écriture -a.b représenterait en fait **non(a.b)**, d'où la nécessité des parenthèses pour imposer une priorité.

Une fois la liste constituée il va falloir la transformer en un arbre syntaxique tout en tenant compte des priorités. Ceci pour ramener au domaine ***expr*** défini. C'est ce que réalisent les prédicats ***arbre***, ***terme***, ***facteur***, ***moins***, ***plus***, ***produit***, ***parenthèse_ouvr*** et ***parenthèse_ferm***.

Ainsi l'arbre relatif à la liste ["a","+","b"] est **ou(booléen("a"),booléen("b"))** et celui du 2ème exemple décrit plus haut: **ou(et(non(booléen("a")),booléen("b")),et(booléen("a"),non(booléen("b"))))**.

La troisième partie va consister à simplifier (si possible) l'expression entrée.

C'est le prédicat ***traduit*** qui en est chargé. Il est défini de manière <u>récursive</u> et utilise le prédicat ***transforme*** tant qu'il y a réellement transformation, c'est-à-dire tant que l'expression obtenue est différente de l'expression à transformer.

La majorité des règles de transformation (simplification) a été énoncée. Elles peuvent paraître nombreuses mais elles sont très souvent dédoublées du fait de la commutativité des opérations logiques.

L'arbre syntaxique éventuellement simplifié sera affiché à l'écran.

• L'opération suivante va être la conversion de l'arbre simplifié en une chaîne de caractères afin, là encore, de rendre le programme convivial. C'est le prédicat ***convertit*** qui assure cette fonction. Cependant, la chaîne obtenue ne sera pas tout à fait affichée telle quelle. Pour plus de clarté, les prédicats ***écrire_car*** et ***écrire_chaîne*** s'en chargeront. Si le résultat contient un **NON** s'appliquant à une seule variable logique alors l'écriture habituelle sera utilisée (lettre surlignée). Par contre si le **NON** s'applique à toute une expression l'écriture employée sera **non(expression)**, par exemple **non(a.b)**.

Le dernier point est la transformation de l'arbre simplifié en un arbre n'utilisant que des **NAND à 2 entrées** (dans la mesure où cela est nécessaire).

La transformation est bâtie sur le même principe que la simplification. Ce sont les prédicats ***traduit_nand*** et ***transforme_nand*** qui règlent ce problème.

Pour utiliser cette partie du programme, choisir l'option **Run** puis, après Goal, taper ***run***. Deux exemples sont traités <u>encadrés 5 et 6</u>.

Jean-Paul POSTEC

```
Dialog
Goal: évalue(ou(booléen(A),b
ooléen(B)),S)
A=0, B=0, S=0
A=0, B=1, S=1
A=1, B=0, S=1
A=1, B=1, S=1
4 Solutions
```

**Encadré 1**

```
Dialog
True
Goal: évalue(et(booléen(A),b
ooléen(B)),S)
A=0, B=0, S=0
A=0, B=1, S=0
A=1, B=0, S=0
A=1, B=1, S=1
4 Solutions
```

**Encadré 2**

```
Dialog
True
Goal: évalue(ou(booléen(A),e
t(booléen(B),booléen(C))),S)
A=0, B=0, C=0, S=0
A=0, B=0, C=1, S=0
A=0, B=1, C=0, S=0
A=0, B=1, C=1, S=1
A=1, B=0, C=0, S=1
A=1, B=0, C=1, S=1
A=1, B=1, C=0, S=1
A=1, B=1, C=1, S=1
8 Solutions
```

**Encadré 3**

```
Dialog
True
Goal: évalue(ou(booléen(A),e
t(non(booléen(A)),booléen(B)
)),S)
A=0, B=0, S=0
A=0, B=1, S=1
A=1, B=0, S=1
A=1, B=1, S=1
4 Solutions
```

**Encadré 4**

(suite de la page 6)

Concernant Turbo-FORTH, nous ne pouvons garantir aucun délai de parution. Turbo-FORTH est encore en chantier. Turbo-FORTH est avant tout une notion avant d'être un produit. Le principe est le suivant: en partant du FORTH 83-Standard de Laxen et Perry, que peut-on définir pour améliorer les performances du langage en respectant l'esprit TURBO de BORLAND.

Actuellement, Turbo-FORTH dispose des fonctionnalités suivantes:
- INCLUDE réentrant (jusqu'à 20 fichiers, [illegible] en standard) et interactif.
- gestion MSDOS complète (CHDIR, MKDIR, RMDIR, REN, DEL, etc...).
- appel de programmes .COM extérieurs à Turbo-FORTH.
- gestion de toute la mémoire vive (256, 512 ou 640K).

suite page 14

```
------------------------------EXPRESSION LOGIQUE------------------------------04:14:35 PM
Entrez l'expression sous forme de somme logique :
S = a+(-a).(-b)
```

```
------------------------------TRAITEMENTS------------------------------
Arbre Syntaxique :
ou(booléen("a"),et(non(booléen("a")),non(booléen("b"))))

Simplication de l'arbre :
ou(booléen("a"),non(booléen("b")))

Expression simplifiée :

S = a + b̄

Traduction NAND à 2 entrées :
nand(nand(booléen("a"),booléen("a")),booléen("b"))
```

**Encadré 5**

```
/***************************************************************************/
/* PROGRAMME PERMETTANT DE TRAITER UNE EXPRESSION LOGIQUE (simplification,  */
/* traduction en portes NAND à 2 entrées) OU D'EVALUER UNE EXPRESSION (table*/
/* de vérité).                                                               */
/*       Ecrit en TURBO-PROLOG par Jean-Paul POSTEC     NANTES Mai 1987      */
/***************************************************************************/

domains
   expr=booléen(string); ou(expr,expr); et(expr,expr);
        nand(expr,expr); nor(expr,expr); non(expr)
   liste=string*

predicates
   évalue(expr,integer)
   expr_liste(string,liste)
   arbre(liste,liste,expr)
   terme(liste,liste,expr)
   facteur(liste,liste,expr)
   moins(liste,liste)
   plus(liste,liste)
   produit(liste,liste)
   parenthèse_ouvr(liste,liste)
   parenthèse_ferm(liste,liste)
   transforme(expr,expr)
   transforme_nand(expr,expr)
   traduit(expr,expr)
   traduit_nand(expr,expr)
   égal(expr,expr)
   différent(expr,expr)
   convertit(expr,string)
   écrire_chaîne(string)
   écrire_car(string)
   run

clauses
/***************************************************************************/
/* Ces clauses permettent l'évaluation d'une expression logique et grâce au */
/* non-déterministe de donner la table de vérité.                           */
/***************************************************************************/

   évalue(booléen("0"),0).
   évalue(booléen("1"),1).

   évalue(ou(E1,E2),X) if évalue(E1,X1), évalue(E2,X2), bitor(X1,X2,X).

   évalue(et(E1,E2),X) if évalue(E1,X1), évalue(E2,X2), bitand(X1,X2,X).
```

```
évalue(nand(E1,E2),X) if évalue(et(E1,E2),Y), bitnot(Y,Z), X=Z+2.

évalue(nor(E1,E2),X) if évalue(ou(E1,E2),Y), bitnot(Y,Z), X=Z+2.

évalue(non(E),X) if évalue(E,Y), bitnot(Y,Z), X=Z+2.

/*************************************************************************/
/*    Clauses de transformation d'une expression sous forme d'une liste   */
/*************************************************************************/

expr_liste(Expr,[Terme|Suite]) if fronttoken(Expr,Terme,Reste), !,
                                  expr_liste(Reste,Suite).
expr_liste(_,[]).

/*************************************************************************/
/* Clauses de transformation d'une liste en arbre binaire syntaxique tout */
/* en tenant compte des règles de priorité des opérations et parenthèses. */
/*************************************************************************/

arbre(X,Y,ou(E1,E2)) if terme(X,Z,E1), plus(Z,W), !, arbre(W,Y,E2).
arbre(X,Y,A) if terme(X,Y,A).

terme(X,Y,et(E1,E2)) if facteur(X,Z,E1), produit(Z,W), !, terme(W,Y,E2).
terme(X,Y,non(E)) if moins(X,W), !, terme(W,Y,E).
terme(X,Y,A) if facteur(X,Y,A).

facteur(X,Y,A) if parenthèse_ouvr(X,Z), arbre(Z,W,A), !,
                  parenthèse_ferm(W,Y).
facteur([H|X],X,booléen(H)).

moins(["-"|X],X).

plus(["+"|X],X).

produit(["."|X],X).

parenthèse_ouvr(["("|X],X).

parenthèse_ferm([")"|X],X).

/*************************************************************************/
/* Clause permettant différents traitements d'une expression logique entrée */
/* sous forme de somme. Ces traitements sont :                            */
/* - la sortie de l'arbre binaire syntaxique                              */
/* - la simplification (éventuelle) de l'arbre syntaxique                 */
/* - la traduction (éventuelle) avec des NAND à 2 entrées.                */
/*************************************************************************/

run if makewindow(1,2,7,"EXPRESSION LOGIQUE",0,0,5,80),
       makewindow(2,31,5,"TRAITEMENTS",5,0,20,80), shiftwindow(1),
       write("Entrez l'expression sous forme de somme logique :\n"),
       write("S = "),readln(Expr), expr_liste(Expr,Liste),
       arbre(Liste,_,Arbre), shiftwindow(2),
       write("Arbre Syntaxique :\n",Arbre), nl, nl,
       traduit(Arbre,S),
       write("Simplication de l'arbre :\n",S), nl, nl,
       convertit(S,Expr_simpl),
       write("Expression simplifiée :\n"), nl,
       write("S = "),écrire_chaîne(Expr_simpl), nl, nl,
       traduit_nand(S,N),
       write("Traduction NAND à 2 entrées :\n",N), nl, fail
       ; readchar(_).

/*************************************************************************/
/*           Règles de simplification de la logique booléenne             */
/*************************************************************************/

transforme(booléen(X),booléen(X)).
```

```
transforme(ou(E,non(E)),booléen("1")).
transforme(ou(non(E),E),booléen("1")).
transforme(ou(_,booléen("1")),booléen("1")).
transforme(ou(booléen("1"),_),booléen("1")).
transforme(ou(E,booléen("0")),X) if transforme(E,X).
transforme(ou(booléen("0"),E),X) if transforme(E,X).
transforme(ou(E1,ou(E1,E2)),ou(X,Y)) if transforme(E1,X), transforme(E2,Y).
transforme(ou(E1,ou(E2,E1)),ou(X,Y)) if transforme(E1,X), transforme(E2,Y).
transforme(ou(E1,et(non(E1),E2)),ou(X,Y)) if transforme(E1,X),
                                              transforme(E2,Y).
transforme(ou(E1,et(E2,non(E1))),ou(X,Y)) if transforme(E1,X),
                                              transforme(E2,Y).
transforme(ou(et(non(E1),E2),E1),ou(X,Y)) if transforme(E1,X),
                                              transforme(E2,Y).
transforme(ou(et(E2,non(E1)),E1),ou(X,Y)) if transforme(E1,X),
                                              transforme(E2,Y).

transforme(ou(non(E1),et(E1,E2)),ou(non(X),Y)) if transforme(E1,X),
                                                   transforme(E2,Y).
transforme(ou(non(E1),et(E2,E1)),ou(non(X),Y)) if transforme(E1,X),
                                                   transforme(E2,Y).
transforme(ou(et(E1,E2),non(E1)),ou(non(X),Y)) if transforme(E1,X),
                                                   transforme(E2,Y).
transforme(ou(et(E2,E1),non(E1)),ou(non(X),Y)) if transforme(E1,X),
                                                   transforme(E2,Y).
transforme(ou(E1,et(E1,_)),X) if transforme(E1,X).
transforme(ou(E1,et(_,E1)),X) if transforme(E1,X).
transforme(ou(et(E1,_),E1),X) if transforme(E1,X).
transforme(ou(et(_,E1),E1),X) if transforme(E1,X).
transforme(ou(E,E),X) if transforme(E,X).
transforme(ou(et(E1,E2),et(E1,E3)),et(X,ou(Y,Z))) if transforme(E1,X),
                                                      transforme(E2,Y),
                                                      transforme(E3,Z).
transforme(ou(et(E1,E2),et(E3,E1)),et(X,ou(Y,Z))) if transforme(E1,X),
                                                      transforme(E2,Y),
                                                      transforme(E3,Z).
transforme(ou(et(E1,E2),et(E2,E3)),et(X,ou(Y,Z))) if transforme(E2,X),
                                                      transforme(E1,Y),
                                                      transforme(E3,Z).
transforme(ou(et(E1,E2),et(E3,E2)),et(X,ou(Y,Z))) if transforme(E2,X),
                                                      transforme(E1,Y),
                                                      transforme(E3,Z).
transforme(ou(E1,ou(E2,E3)),ou(ou(X,Y),Z)) if transforme(E1,X),
                                               transforme(E2,Y),
                                               transforme(E3,Z).
transforme(ou(ou(E1,E2),E3),ou(X,Y)) if transforme(ou(E1,E3),X),
                                        transforme(E2,Y),
                                        différent(ou(X,Y),ou(ou(E1,E3),Y)).
transforme(ou(E1,E2),ou(X,Y)) if transforme(E1,X), transforme(E2,Y).

transforme(et(E,non(E)),booléen("0")).
transforme(et(non(E),E),booléen("0")).
transforme(et(E,E),X) if transforme(E,X).
transforme(et(E,booléen("1")),X) if transforme(E,X).
transforme(et(booléen("1"),E),X) if transforme(E,X).
transforme(et(_,booléen("0")),booléen("0")).
transforme(et(booléen("0"),_),booléen("0")).
transforme(et(E1,et(E1,E2)),et(X,Y)) if transforme(E1,X), transforme(E2,Y).
transforme(et(E1,et(E2,E1)),et(X,Y)) if transforme(E1,X), transforme(E2,Y).
transforme(et(E1,E2),X) if égal(E1,E2), transforme(E1,X).
transforme(et(E1,et(E2,E3)),et(et(X,Y),Z)) if transforme(E1,X),
                                               transforme(E2,Y),
                                               transforme(E3,Z).
transforme(et(et(E1,E2),E3),et(X,Y)) if transforme(et(E1,E3),X),
                                        transforme(E2,Y),
                                        différent(et(X,Y),et(et(E1,E3),Y)).
transforme(et(E1,E2),et(X,Y)) if transforme(E1,X), transforme(E2,Y).

transforme(non(booléen("0")),booléen("1")).
transforme(non(booléen("1")),booléen("0")).
transforme(non(non(E)),X) if transforme(E,X).
transforme(non(E),non(X)) if transforme(E,X).
```

```
/****************************************************************************/
/*     Règles de transformation des circuits en portes NAND à 2 entrées     */
/****************************************************************************/

   transforme_nand(booléen(X),booléen(X)).
   transforme_nand(ou(E1,E2),nand(nand(X,X),nand(Y,Y))) if
                                           transforme_nand(E1,X),
                                           transforme_nand(E2,Y).
   transforme_nand(et(E1,E2),nand(nand(X,Y),nand(X,Y))) if
                                           transforme_nand(E1,X),
                                           transforme_nand(E2,Y).
   transforme_nand(nand(E1,E2),nand(X,Y)) if transforme_nand(E1,X),
                                           transforme_nand(E2,Y).
   transforme_nand(nand(nand(E,E),nand(E,E)),X) if transforme_nand(E,X).
   transforme_nand(non(E),nand(X,X)) if transforme_nand(E,X).

   traduit(E1,E2) if transforme(E1,A), différent(A,E1), !, traduit(A,E2).
   traduit(E,E).

   traduit_nand(E1,E2) if transforme_nand(E1,A), différent(A,E1), !,
                          traduit_nand(A,E2).
   traduit_nand(E,E).

   égal(X,X).

   différent(X,Y) if not(égal(X,Y)).

/****************************************************************************/
/* Clauses convertissant une expression sous forme de chaîne, cela est in-  */
/* dispensable pour l'affichage de l'expression simplifiée.                 */
/****************************************************************************/

   convertit(booléen(X),X).

   convertit(ou(E1,E2),Chaîne) if convertit(E1,Ch1), convertit(E2,Ch2),
                           concat(Ch1,"+",Ch3), concat(Ch3,Ch2,Chaîne).

   convertit(et(E1,ou(E2,E3)),Chaîne) if convertit(E1,Ch1),
                                   convertit(ou(E2,E3),Ch2),
                                   concat(Ch1,".(",Ch3),
                                   concat(Ch3,Ch2,Ch4),
                                   concat(Ch4,")",Chaîne), !.
   convertit(et(ou(E1,E2),E3),Chaîne) if convertit(E3,Ch1),
                                   convertit(ou(E1,E2),Ch2),
                                   concat("(",Ch2,Ch3),
                                   concat(Ch3,").",Ch4),
                                   concat(Ch4,Ch1,Chaîne), !.
   convertit(et(E1,E2),Chaîne) if convertit(E1,Ch1), convertit(E2,Ch2),
                           concat(Ch1,".",Ch3), concat(Ch3,Ch2,Chaîne).

   convertit(non(booléen(X)),Ch) if concat("┐",X,Ch), !.
   convertit(non(E),Ch) if convertit(E,Ch1), concat("non(",Ch1,Ch2),
                        concat(Ch2,")",Ch).

/****************************************************************************/
/*     Clauses utilisées par l'affichage de l'expression simplifiée         */
/****************************************************************************/

   écrire_car("┐") if cursor(X,Y), X1=X-1, cursor(X1,Y),
                     write("_"), cursor(X,Y), !.
   écrire_car("+") if write(" + "), !.
   écrire_car(X) if write(X).

   écrire_chaîne("") if !.
   écrire_chaîne(Chaîne) if fronttoken(Chaîne,Premier,Suite),
                          écrire_car(Premier), écrire_chaîne(Suite).
```

## INTRODUCTION

Il est possible de réaliser des graphiques avec le FORTH F83-STANDARD. Pour cela il faut posséder un micro-ordinateur IBM-PC ou compatible, une carte HERCULES et le FORTH-83 (F83.COM).

Le logiciel qui accompagne la carte HERCULES comprend différents modules qui permettent de travailler à partir des langages suivants: assembleur, basic, pascal et fortran. Tous ces modules transitent par le module INT10.COM qui est une extension du programme d'interruption INT 10H du BIOS d'IBM. En effet ce dernier utilise le vecteur d'interruption INT 10H pour gérer l'écran du PC. Les concepteurs de la carte HERCULES ont complété ce programme pour lui adjoindre leurs propres fonctions graphiques.

Il est donc nécessaire de charger le programme INT10.COM après la mise sous tension du micro-ordinateur afin d'accéder aux fonctions graphiques de la carte HERCULES. De plus la commande HGC.COM doit-être exécutée avec l'option FULL. Cette commande permet d'utiliser la carte HERCULES avec toutes ses options: 2 pages graphiques et un "buffer" de texte. Pour de plus amples informations reportez-vous au manuel HERCULES.

## DESCRIPTION DES FONCTIONS

Quinze fonctions sont disponibles, qui peuvent-être réparties en deux groupes: fonctions d'attribut et fonctions de traçage.

### *FONCTIONS D'ATTRIBUT*

| | | |
|---|---|---|
| -- GMODE | : | PLACE L'ECRAN EN MODE GRAPHIQUE ET AFFICHE LA PAGE 0 |
| -- TMODE | : | PLACE L'ECRAN EN MODE ALPHANUMERIQUE |
| -- GPAGE | : | SELECTION DE LA PAGE GRAPHIQUE DE TRAVAIL |
| -- AFFICHE | : | SELECTION DE LA PAGE GRAPHIQUE A AFFICHER |
| -- INTENSITE | : | SELECTION DU NIVEAU D'INTENSITE DU TRACE |

L'écran est défini de 0 à 719 points en horizontal et de 0 à 347 points en vertical. Le point 0,0 se trouve au coin haut gauche de l'écran.

### *FONCTIONS DE TRACAGE*

| | | |
|---|---|---|
| -- EFFACE | : | EFFACE ENTIEREMENT LA PAGE GRAPHIQUE COURANTE |
| -- POINT | : | TRACE UN POINT A LA POSITION X,Y |
| -- LECTPT | : | LECTURE DU NIVEAU D'INTENSITE DU POINT A LA POSITION X,Y |
| -- DEPLACE | : | DEPLACE UN CURSEUR FICTIF A LA POSITION X,Y |
| -- LIGNE | : | TRACE UNE LIGNE DE LA POSITION DU CURSEUR A LA POSITION X,Y |
| -- BLKFILL | : | TRACE UN RECTANGLE DE TAILLE VARIABLE A LA POSITION X,Y |
| -- TEXT | : | TRACE UN CARACTERE A LA POSITION X,Y |
| -- ARC | : | TRACE UN QUART DE CERCLE AVEC UN RAYON R CENTRE A LA POSITION X,Y |
| -- CERCLE | : | TRACE UN CERCLE AVEC UN RAYON R CENTRE A LA POSITION X,Y |
| -- REMPLI | : | REMPLISSAGE DE LA SURFACE D'UN POLYGONE DONT LA POSITION D'UN POINT EST DONNEE. |

## DESCRIPTION DES DEFINITIONS FORTH

L'utilisateur accède à ces fonctions par l'interruption logicielle 10H. Le code spécifiant la fonction est chargé dans le registre AH du microprocesseur et les registres DI, BP, BX, CX ou AL reçoivent les paramètres nécessaires à l'exécution de la fonction.

Les codes fonctions s'échelonnent de 40H à 4EH. Rien de plus simple donc, que d'écrire des routines en assembleur (à partir du FORTH) qui permettent d'exécuter toutes les fonctions graphiques. C'est ce qu'indiquent les 7 écrans du "listing" ci-après.

Le programme INT10.COM sauvegarde uniquement les registres CS, IP et les "flags" d'états. Par conséquent il faut impérativement sauvegarder les registres utilisés pour la gestion du FORTH. Ce sont les registres BP et SI utilisés respectivement comme pointeur de pile retour et pointeur d'interprétation.

## UTILISATION

Une application graphique doit toujours débuter par un GMODE et se terminer par un TMODE (voir les exemples ESSAIn). Pour visualiser un graphique il faut maintenir l'écran en mode graphique. Le programme d'application doit donc contenir une routine qui place le programme en attente de la frappe d'une touche du clavier. Cette action est réalisée par la définition TOUCHE qui utilise l'interruption logicielle INT 16H.

Quand on débute une application il est nécessaire d'effacer auparavant la page de travail aux risques de voir apparaître des caractères cabalistiques.

La page 1 peut servir de page mémoire. C'est à dire contenir un tracé qui peut-être rappelé pour affichage à tout instant, sans modifier son contenu. Par contre il n'en est pas de même pour la page 0 qui, si elle est rappelée pour affichage contient ,en plus du tracé, des signes cabalistiques (exemple: ESSAI1 ESSAI6). Ceci est dû à la présence du "buffer" de texte (4 k octets) qui est situé au bas de la page 0.

Pour remédier à cet inconvénient il faudrait sauvegarder la page graphique dans un fichier disque ou dans un "buffer" de la mémoire centrale. C'est une solution à laquelle je travaille.

Cette interface logicielle n'est qu'un outil de base et doit-être complétée pour être plus performante. Il manque par exemple la définition pour afficher une chaîne de caractères alphanumériques, le tracé d'une ellipse, etc...

(suite de la page 8)
- éditeur plein écran pour fichiers ASCII appelé depuis FORTH en tant que programme externe. Le positionnement dans le texte sera du style:

COMMAND EDIT texte.fth ligne colonne

où EDIT est le nom du fichier EDIT.COM et les paramètres suivent.
- exécution depuis MSDOS d'un module précompilé par Turbo-FORTH. Exemple:

TURBO WORDS BYE > WORDS.DOC

lance TURBO depuis MSDOS, exécute WORDS puis retourne à MSDOS et écrit le résultat de l'action de WORDS dans un fichier ASCII nommé WORDS.DOC. Imaginez l'utilité de ce processus s'il est exploité dans un fichier .BAT.

Sont en chantier les modules internes:
- redirection de l'affichage vers un fichier ASCII. Exemple:

```
ALTERNATE: fichier.doc
ALTERNE ON WORDS ALTERNE OFF
```

crée un fichier, redirige l'affichage écran vers ce fichier et enregistre toutes les actions Turbo-FORTH dans ce fichier tant que ALTERNE ne passe pas à OFF.

Sont en chantier des modules externes:
- fonctions graphiques (diffusées dans le présent numéro)
- variables locales
- méta-compilateur générant du code dans le segment mémoire compris entre 64k et 128k, ce qui permet la méta-génération d'un fichier .COM de 64k et non plus limité à environ 15k comme actuellement.
- et d'autres idées...

Naturellement, toutes ces fonctions seront développées dans JEDI afin de vous permettre d'aligner toute ou partie de F83 Laxen et Perry à Turbo-FORTH, le but étant de permettre une compatibilité aussi grande que possible des programmes entre F83 et Turbo-FORTH.

Marc PETREMANN
Secrétaire de l'Association

COMPLEMENT A LA LETTRE DE MR BARONE:

Bonjour, suite au numéro 38 et à ma lettre que vous avez publiée, je vous fais parvenir ce courrier qui pourra permettre éventuellement de meubler une page ou deux d'un futur N° de JEDI. Je n'ai pas de traitement de texte sur mon PC (quand je dois écrire j'utilise un Macintosh) j'espère que vous pourrez récupérer ce fichier tapé avec l'éditeur de TURBOPASCAL.

Tout d'abord quelques mots de présentation : je suis chercheur au CNRS, chimiste de formation, mais ayant bifurqué vers l'informatique lorsqu'on m'a proposé comme sujet de thèse de résoudre les problèmes de synthèse de molécules au moyen de l'ordinateur. On donne une molécule à l'ordinateur et celui-ci, au moyen des réactions chimiques qu'il a en mémoire, doit faire des propositions de synthèse. L'approche est dite rétrosynthétique : on donne la molécule à synthétiser (molécule cible) à

suite page 17

```
   1
\ ROUTINES D'ACCES POUR LA CARTE HERCULE              12/6/87

CODE GMODE (S --)
64 # AH MOV BP PUSH SI PUSH 16 INT
SI POP BP POP NEXT C;

CODE TMODE (S --)
65 # AH MOV BP PUSH SI PUSH 16 INT
SI POP BP POP NEXT C;

CODE CERCLE (S x y r --)
BX POP DX POP DI POP
BP PUSH SI PUSH DX BP MOV
77 # AH MOV 16 INT SI POP BP POP NEXT C;
```

```
   2
\ SUITE HERCULE                                       18/5/87
CODE EFFACE (S --)
66 # AH MOV BP PUSH SI PUSH 16 INT
SI POP BP POP NEXT C;

CODE GPAGE (S n --)
AX POP 67 # AH MOV BP PUSH SI PUSH 16 INT
SI POP BP POP NEXT C;

CODE AFFICHE (S n --)
AX POP 69 # AH MOV BP PUSH SI PUSH 16 INT
SI POP BP POP NEXT C;

CODE INTENSITE (S n --)
AX POP 68 # AH MOV BP PUSH SI PUSH 16 INT
SI POP BP POP NEXT C;
```

```
   3
\ SUITE HERCULE                                       12/6/87
CODE TOUCHE (S --)
0 # AH MOV BP PUSH SI PUSH 22 INT
SI POP BP POP NEXT C;

CODE LECTPT (S x y --n)
BX POP DI POP
BP PUSH SI PUSH
BX BP MOV
71 # AH MOV 16 INT
SI POP BP POP 0 # AH AND AX PUSH
NEXT C;
```

```
   4
\ SUITE HERCULE                                       12/6/87
CODE ARC (S x y r c --)
AX POP BX POP
DX POP
DI POP
BP PUSH SI PUSH
DX BP MOV
76 # AH MOV
16 INT SI POP BP POP NEXT C;

CODE BLKFILL (S x y l h --)
BX POP CX POP AX POP DI POP
BP PUSH SI PUSH AX BP MOV
74 # AH MOV
16 INT SI POP BP POP NEXT C;
```

```
   5
\ SUITE HERCULE                                       12/6/87
CODE LIGNE (S x y --)
BX POP DI POP
BP PUSH SI PUSH BX BP MOV
73 # AH MOV 16 INT
SI POP BP POP  NEXT C;

CODE DEPLACE (S x y  --)
BX POP DI POP
BP PUSH SI PUSH BX BP MOV
72 # AH MOV 16 INT
SI POP BP POP NEXT C;

CODE REMPLI (S x y --)
AX POP DI POP BP PUSH SI PUSH
AX BP MOV 78 # AH MOV 16 INT SI POP BP POP NEXT C;
```

```
   6
\                                                     12/6/87
CODE TEXT (S x y caract. --)
BX POP AX POP DI POP
BP PUSH SI PUSH AX BP MOV BX AX MOV
75 # AH MOV 16 INT SI POP BP POP NEXT C;

CODE POINT (S x y --)
BX POP DI POP
BP PUSH SI PUSH BX BP MOV
70 # AH MOV 16 INT
SI POP BP POP NEXT C;
```

Forth 83 Model

7

```
\ SUITE HERCULE                                    30/7/87

CODE K (S -- n)
2 [RP] AX MOV 32768 # AX ADD 1PUSH C;
```

8

```
\ EXEMPLES                                         6/09/87
( TRACE UN CERCLE DANS LA PAGE 0 )
: ESSAI1 GMODE EFFACE 300 150 100
  CERCLE TOUCHE TMODE ;
( TRACE UN LIGNE DANS LA PAGE 0 )
: ESSAI2 GMODE EFFACE 0 347 DEPLACE
  719 0 LIGNE TOUCHE TMODE ;
( TRACE UN CERCLE DANS LA PAGE 1 ET L'AFFICHE )
: ESSAI3 GMODE EFFACE 1 GPAGE EFFACE 300 150 120 CERCLE
  1 AFFICHE TOUCHE TMODE ;
( TRACE UN CARACTERE DANS LA PAGE 0 )
: ESSAI4 GMODE EFFACE 100 100 ASCII A TEXT TOUCHE TMODE ;
( TRACE UN POINT DANS LA PAGE 0 )
: ESSAI5 GMODE EFFACE 100 150 POINT TOUCHE TMODE ;
( AFFICHE LA PAGE 0 )
: ESSAI6 GMODE 0 AFFICHE TOUCHE TMODE ;
```

9

```
\ SUITE EXEMPLES                                   6/09/87
( AFFICHE LA PAGE 1 )
: ESSAI7 (S --)
  GMODE 1 AFFICHE TOUCHE TMODE ;
( TRACE UN ARC DE CERCLE DANS LA PAGE 0 )
: ESSAI8 (S --)
  GMODE EFFACE 0 347 521 1 ARC TOUCHE TMODE ;
( TRACE UN RECTANGLE COLORIE )
: ESSAI9 (S --)
  GMODE EFFACE 100 100 200 50 BLKFILL TOUCHE TMODE ;
( TRACE UN CERCLE COLORIE )
: ESSAI10 (S --)
  GMODE EFFACE 300 150 120 CERCLE 300 150 REMPLI TOUCHE
  TMODE ;
```

10

```
\ COMMENTAIRES                                     12/6/87
GMODE     :POSITIONNE LA CARTE EN MODE GRAPHIQUE
TMODE     :POSITIONNE LA CARTE EN MODE TEXTE
GPAGE     :SELECTION DE LA PAGE GRAPHIQUE DE TRAVAIL n= 0 OU 1
AFFICHE   :SELECTION DE LA PAGE GRAPHIQUE A VISUALISEE n= 0 OU 1
INTENSITE :SELECTION DU NIVEAU D'INTENSITE DU SPOT n= 0,1,2
           0= NOIR ; 1= BLANC ; 2= OU EXCLUSIF ENTRE LE NIVEAU
           SELECTIONNE ET LE NIVEAU DU POINT EXISTANT
EFFACE    :EFFACE LA PAGE GRAPHIQUE COURANTE
POINT     :AFFICHE UN POINT AUX COORDONNEES x y
LECTPT    :RECUPERE L'INTENSITE D'UN POINT n=0 OU 1
DEPLACE   :DEPLACE UN CURSEUR IMAGINAIRE AUX COORDONNEES x y
LIGNE     :TRACE UNE DROITE DU CURSEUR AU POINT x y
BLKFIIL   :PEINT UN RECTANGLE DE LARGUEUR 1 ET DE HAUTEUR h
           DONT LE COIN BAS GAUCHE EST AUX COORDONNEES x y .
TEXT      :ECRITURE D'UN CARACTERE ASCII AUX COORDONNEES x y
```

11

```
\ SUITE COMMENTAIRES                               25/7/87
ARC    : TRACE UN QUART DE CERCLE
         CENTRE : x y
         RAYON  : r
         CADRAN : q     1= 0° à 90°      2= 90° à 180°
                      3= 180° à 270°   4= 270° à 360°
CERCLE : TRACE UN CERCLE
         CENTRE : x y
         RAYON  : r
REMPLI : PEINT LA SURFACE DELIMITEE PAR UN CONTOUR FERME
         DONT UN DES POINTS EST FOURNI PAR LES COORDONNEES x y
TOUCHE : ATTENTE EXPECTATIVE DE LA FRAPPE D'UNE TOUCHE.
         LA VALEUR DU CARACTERE N'EST PAS CONSERVEE.
K      : RECUPERE LA LIMITE D'UNE BOUCLE DO
```

0

```
                                                   10/9/87
################################################################
#                                                              #
#                                                              #
#                                                              #
#         INTERFACE LOGICIELLE POUR CARTE HERCULES             #
#                                                              #
#                                                              #
#                                                              #
#                                                              #
#                                                              #
#                                                              #
#                                                              #
#                                                              #
#                                                              #
################################################################
```

Forth 83 Model

(suite de la page 14)
l'ordinateur, celui-ci propose différents précurseurs, puis ces précurseurs deviennnent de nouvelles cibles et le processus est répété jusqu'à ce qu'on aboutisse à des solutions intéressantes. Le programme est interactif : le chimiste sélectionne les intermédiaires intéressants (l'ordinateur fait des erreurs) et guide la rétrosynthèse en choisissant les nouvelles cibles au fur et à mesure. On peut schématiser cette démarche par un arbre :

```
cible        ┌────────> M <──────┐
             │                   │
          ┌─M1─┐        M2       M3
          │    │        │        │
          M4   M5       M6       M7
       etc...
```

On peut synthétiser M à partir de M1 ou M2 ou M3

Si vous trouvez le sujet intéressant je pourrais entrer dans les détails de ce programme : codage des molécules, codages des connaissances (réactions) etc.

Mais je vais plutôt aborder le côté informatique. A l'origine, débutant en programmation, j'ai appris ce qui s'enseignait à l'époque (en 1970) à l'université: le FORTRAN et une première version (1973) puis une seconde (1976) tournaient sur un ordinateur de 16 Kmots (de 16bits) [un IBM 11/30 qui occupait une pièce entière, réfrigérée, le must des années 65]. Puis vint Apple (64 Ko sur un bureau, le pied!) et une adaptation BASIC de notre programme. Mais ce n'était pas suffisant pour développer un système expert digne de ce nom. 1984: achat d'un Macintosh et recherche d'un langage de programmation intelligent pour la quatrième version du programme (on a de la suite dans les idées au CNRS!) entre temps, la version BASIC avait été commercialisée par une société qui l'avait mis sur un IBM/PC, et malgré un prix de vente incroyable: 78 000 Francs HT (mais oui) 5 ou 6 sociétés en avaient fait l'acquisition : ce devait donc être un bon programme!).

Donc recherche d'un langage de programmation génial:
- BASIC: (celui de l'époque, en 84 il n'y avait pas de compilateurs) pas assez rapide.
- PASCAL: intéressant mais je trouvais la syntaxe un peu trop lourde.
- C: je ne connaissais pas encore et rien ou presque sur le Mac.
- APL: hermétique et indisponible sur Mac à l'époque.

Il faut dire qu'au début sur Mac, il n'y avait pas beaucoup de langages: il fallait développer en Pascal sur LISA, et notre laboratoire n'était pas assez riche pour s'en payer un. Le Macintosh avait été choisi, car Apple faisait des prix: 40% de réduction pour les développeurs, et la souris était l'outil idéal pour dessiner des molécules sur un écran.

Suite du roman feuilleton: recherche d'un langage intelligent ET disponible sur Mac. Fin 1983, dans Micro systèmes était paru une série d'articles sur FORTH. Dans cette même revue découverte de JEDI, c'est pourquoi je suis ici aujourd'hui. FORTH paraissait intéressant par son extensibilité et rapide (en fait il ne l'est pas autant que je l'espérais). Or MacFORTH était, avec le Basic interprété de Microsoft, le seul langage permettant de développer directement sur Mac. Le choix du langage était donc trouvé.

Fin 1987: une version de notre logiciel tourne en FORTH sur Macintosh et maintenant il nous faut l'adapter sur IBM/PC, car dans l'industrie on trouve plus de PC et compatibles que des Mac. D'où nouveau problème: quel FORTH choisir? Et là pas facile à trouver, car ce n'est pas le langage de programmation le plus répandu! Comme je l'indiquais, celui de LMI semble le mieux (ou plutôt le seul que j'ai pu trouver). Je ne connais pas celui de Laxen et Perry. Cependant, celui de LMI paraît très complet du point de vue graphique. On peut travailler en CGA, EGA et HERCULES. PARENTHESE: dès qu'on veut faire du graphisme sur IBM/PC ce n'est pas simple (surtout si on veut écrire un programme général qui puisse tourner sur les modes graphiques les plus répandus; TURBO et QUICK BASICS n'ont pas prévu la carte HERCULES, en C il faut se payer des bibliothèques qui ne sont pas bon marché, idem en FORTRAN, peut-être TURBO-PASCAL ? FIN DE LA PARENTHESE.

De plus le FORTH de LMI grâce au NATIVE CODE OPTIMIZER semble apporter une rapidité qui fait défaut au FORTH. Quand au prix, celui pratiqué par Micro Sigma, qui le commercialise est très élevé. Il vaut mieux commander aux Etats Unis, non pas chez LMI, mais à l'adresse suivante:

PROGRAMMER'S CONNECTION , 136 Sunnyside Street
HARTVILLE, OHIO 44 632. USA. Tel : (216) 877 37 81.
PC Forth+ , NATIVE CODE ... et les routines graphiques pour moins de 500 $.

Vu la baisse du dollar, c'est intéressant. Cette boîte vend de nombreux logiciels, les personnes intéressées peuvent demander un catalogue. Les prix sont intéressants.

En conclusion et après avoir lu le dernier numéro de JEDI, j'attends avec impatience le TURBO FORTH, qui paraît attrayant pour ses possibilités graphiques, par un éditeur plein écran et par les variables locales pour le passage d'arguments.

Faudrait prévoir un compilateur en code natif, pour être très rapide et abandonner la notation polonaise inversée, à ce moment là, le FORTH sera un langage (presque) génial (même si ça ne ressemble plus beaucoup au FORTH).

Cette dernière phrase me fait penser que j'avais envie de dire quelques mots les langages de programmation: après avoir sélectionné MacFORTH pour développer notre système expert, j'ai continué à chercher LE langage idéal. Bien sûr il n'existe pas, tout dépend de ce que l'on veut faire. Personnellement je cherche un langage: extensible comme FORTH, simple comme BASIC, rapide comme ASSEMBLEUR.

Qui écrira ce langage ? En lisant JEDI, j'ai cru trouver avec LPB les deux derniers points. Malheureusement, son auteur semble avoir quelques problèmes. Dommage car c'est un langage original. En lisant Micro Systèmes, je me suis intéresssé à MAGIC/L: langage très chouette, mais après avoir reçu la doc des Etats Unis, pas de possibilités graphiques, rapide comme un PASCAL UCSD, c'est-à-dire, pas très, dommage car proche du FORTH et du PASCAL.

En ce moment j'apprends C, qui est très bien. Après j'étudierai PROLOG en détail, pour essayer de manipuler des molécules avec. Reste encore LISP, SMALL TALK.

Mon Dieu, que c'est dur la programmation! Je refuse d'apprendre l'assembleur: je ne comprends pas comment, en 1987, on soit encore obligé d'utiliser un tel langage pour avoir un programme très rapide et qu'il n'existe pas un langage simple qui une fois compilé donne la vitesse de l'assembleur! Si il y a LPB, mais voir plus haut. Dommage, c'était peut-être le langage du siècle.

La solution viendra peut-être du microprocesseur NOVIX: pas besoin de programmer en assembleur mais en FORTH! A propos de NOVIX, pour des raisons trop longues à expliquer, j'en ai un exemplaire. Si une personne est intéressée il est à vendre. A propos de ce microprocesseur, il serait bien que JEDI puisse nous tenir au courant de son développement et de ses possibilités.

Qui construira une carte équipée de NOVIX enfichable dans un compatible? On aurait alors une machine vraiment formidable!

NOUVELLE REPONSE DU SECRETAIRE (...disons des éclaircissements):

Nous avons ici une démarche de développeur type passionné par son travail, mais ne sachant pas quelle voie choisir. Qui n'a pas connu ces problèmes me jette le premier clavier...

Il est toujours angoissant de se lancer sur un projet ambitieux avec des moyens dont on n'est pas sûr des performances. Les exigences de la technique ont d'une

certaine manière forgé les outils permettant de maîtriser cette technique. Autrefois, les astronomes grands demandeurs de calculs, utilisaient des outils mathématiques qu'ils avaient eux-même inventé (logarithmes, trigonométrie, calcul intégral...). Cette progression ne s'est pas faite en une seule génération. Entre EUCLIDE et EINSTEIN (qui était physicien, mais avait comme assistant mathématicien le futur inventeur de BASIC...), des milliers de "bidouilleurs" ont élaboré une kyrielle de moyens et de méthodes pour analyser et comprendre la nature du monde, et ce n'est pas fini: les fractals, la théorie des catastrophes, la topologie, les noeuds, la géométrie non-euclidienne, etc...

Il est donc normal que les machines chargées de traiter l'information suivent une évolution identique. On dira que la fonction crée l'organe, or, à preuve du contraire, un organe est toujours très spécialisé. Respectez ce grand principe:

- ne construisez pas des moissonneuses batteuses-lieuses-faucheuses-emboiteuses qui ne serviraient qu'à la récolte du jujube, utilisables seulement dans les pleines septentrionales de l'Inde et seulement 15 jours par an après le solstice d'été (on l'a déjà sortie dans TOURS DE FORTH, écrit par moi-même et ROUSSEAU, ed Eyrolles).

L'idéal, à mon avis, est de disposer de plusieurs langages et programmes et de les faire communiquer ensemble. Turbo-FORTH appelle un module FORTRAN, revient ensuite au DOS, le fichier BAT passe ensuite en Turbo-PROLOG qui exploite un fichier généré par dBASE III, etc... Si, c'est possible! L'exemple est un peu excessif, mais devient de plus en plus fréquent en programmation.

Autrefois, le ZX81 n'avait que le BASIC... Hier, l'APPLE avait le BASIC, PASCAL, FORTH, C, LISP, et d'autres, mais ils ne communiquaient pas. Aujourd'hui, avec 256k au minimum sur un compatible PC de base, on héberge le DOS, un intégré, un programme de traitement de texte et on lance un utilitaire quelconque. Demain, le CD-ROM (peut-être le CD-WRM -Compact-disc Write and Read Memory) avec ses 4Goctets (4Giga-octets) et peut-être plus (on parle déjà de prototypes à 20Goctets) sera tellement interactif que vous aurez l'équivalent de plus de 3000 pages d'encyclopédie avec sélection des corrélats, recherche par mots clés directement dans le texte (exemple: rechercher tous les articles où il est fait référence à l'acide désoxyribonucléique). Demain, des batteries de scanners mettront des archives, des livres et des journaux, des plans (voir le système d'archivage documentaire de LEANORD) en CDROM. Des systèmes d'analyse sémantique et syntaxique pourront élaborer des résumés, voire traduire des articles en langue étrangère.

Et FORTH dans tout çà? Il est destiné à évoluer. Si le NOVIX a fait son apparition, c'est qu'il a un rôle à jouer dans l'informatique de demain. Peut-être n'est-il que le précurseur d'une nouvelle génération de processeurs (on en reparlera dans la série consacrée à la logique). Tout en restant standard, FORTH ne peut rester figé. L'amélioration de ses performances est un élément vital à sa diffusion massive. Rien n'empêche de tirer dans toutes les directions: F83, Turbo-FORTH, carte Delta-BOARD (avec NOVIX 4000), FORTH LMI, etc...

Concernant la proposition de dévelopement du sujet CODAGE DE MOLECULES, je puis avancer avec quasi-certitude qu'un certain nombre d'adhérents bave d'impatience d'en connaître davantage. De même, si vous avez des problèmes pour optimiser votre programme en langage FORTH, parions sur une aide bénévole mais efficace de la part de programmeurs chevronnés en quête d'idée riche (5 oeufs frais au kilo...).

LE SECRETAIRE

COURRIER: Cher ami

Au reçu du dernier N° de la revue je me suis empressé d'implanter Turbo-Forth sur disquette. C'est vraiment sensationnel. Un petit regret toutefois, que la très utile commande view ne puisse être utilisée pour les mots créés par INCLUDE.

Je vous envoie ci-inclus Facpr.txt qui utilise un algorithme de mon crû pour décomposer en produit de facteurs premier un entier double avec une vitesse illustrant, si besoin était, les performances exceptionnelles de Forth.

J'ai trouvé un petit défaut tant dans F83 que dans Turbo-Forth pour utilisation sur PC: la touche "Rubout" (ctrl H) effectue un retour arrière sans effacement. J'ai donc remplacé BS-IN par DEL-IN en position 8 de la control-tabl et tout va bien. Pour F83, plutot que de modifier kernel.blk, j'ai tické (mot affreux mais plus compréhensible pour les forthistes que le poker des basiciens) directement dans la table le cfa de del-in au lieu de celui de bs-in.

J'inclus également un petit bloc EX-BM d'utilitaires divers et en particulier MOTS, analogue à WORDS mais qui affiche en plus l'adresse hexa du lfa de chaque mot ainsi qu'une page d'utilitaires servant de base à la gestion sur disque de fichiers séquentiels.

Le fichier LIT.COM par lequel vous avez probablement entrepris de me lire ,est l'oeuvre de Bruno Tredez , président de la section Ile de France du GUIC (Groupe des Utilisateurs de l'Ibm pc et Compatibles, BP 77, 74600 SEYNOD). J'espère qu'il ne m'en voudra pas de vous l'avoir communiqué.

A titre de contribution à l'association la présente disquette comporte également un fichier de présentation de la pile "heap" de Turbo-pascal, déjà publié par le GUIC et que vous pourrez publier dans la revue Jedi si vous le jugez utile.

A bientot, j'espère. Recevez cher ami l'expression de mes sentiments les meilleurs.

Bernard MORISSEAU
75008 PARIS

## DECOMPOSITION EN FACTEURS PREMIER

par Bernard MORISSEAU

Turbo-FORTH
F83 Laxen et Perry MSDOS et CP/M

```
: DIV 0 BEGIN >R 3DUP >R >R >R MU/MOD ROT R> R> ROT
R> SWAP R> SWAP 0= WHILE
1+ >R -ROT 2DROP R> REPEAT
DUP IF SWAP DUP CR 9 .R SWAP 15 .R
   ELSE DROP THEN >R 2SWAP 2DROP R> ;
: FACPR CR CR ."  Diviseur     Puissance" CR
2 DIV 1+ DIV DROP 5 DIV
BEGIN 2+ DIV 4 + DIV 3DUP DUP *D DU< UNTIL
CR DROP 9 UD.R 1 15 .R CR ;
(            * ESSAIS *        )            CR
743217642. FACPR
985646733. FACPR
```

## UTILITAIRES DIVERS

par Bernard MORISSEAU

Turbo-FORTH
F83 Laxen et Perry MSDOS et CP/M

```
\ Utilitaires divers           11nov87 BM
: H. ( Dépile et affiche en hexa non signé base inchangée)
  BASE DUP @ ROT HEX U. SWAP ! ;
: BASE? BASE DUP @ DUP DECIMAL . SWAP ! ;
: SUIV (S lfa --)
  BEGIN KEY? IF EXIT THEN DUP
  WHILE DUP L>NAME DUP C@ 31 AND
```

```
  ?LINE DUP H. .ID ." : " @ SWAP ! HERE 4 LARGEST
  REPEAT ;
: MOTS ( comme words + affich. lfa des mots)
  CR LMARGIN @ SPACES CONTEXT @ HERE TUCK 8 CMOVE
  4 LARGEST SUIV ;

\ Utilitaires divers            11nov87 BM
: EMPST SP0 @ SP! ; ( vide la pile )
                  : PAD$    ( saisit une chaine et
l'inscrit dans pad )
  PAD DUP 20 32 FILL DUP ." ? "
  20 0 DO  1+ KEY DUP 13 =
          IF DROP LEAVE
          ELSE DUP EMIT OVER C! I 1+ 2 PICK C!
          THEN
      LOOP 32 OVER C! 2DROP ;
: TEXT (S sépar ---)
  PAD 72 32 FILL WORD COUNT PAD SWAP 2DUP SWAP !
  SWAP 1+ SWAP CMOVE ;

\ Utilit. gestion fichiers dsk séquentiels  12nov87 BM
: AFF (S adr long --- )
  0 DO DUP I + C@ . LOOP DROP ;
  VARIABLE NBL   VARIABLE DEP ( N°bloc , déplt )
: &!
  \  oct --  inscrit octet à adr dans bloc et incrém adr
  NBL @ BLOCK DEP DUP >R @ + C!
  R> DUP @ 1024 MOD 1023 =
  IF OFF UPDATE SAVE-BUFFERS 1 NBL TUCK +! @ CAPACITY =
     IF 1 MORE THEN
  ELSE 1 SWAP +! THEN ;
: &NB (.) 0 DO DUP C@ &! 1+ LOOP DROP ;
  \ inscr nbre sur pile dans bloc forme c-caract
: &INP PAD$ PAD COUNT 0
  DO DUP C@ &! 1+ LOOP DROP 13 &! 10 &! ;



\ Décompose en produit de facteurs premiers  12nov87 BM
: DIV (S d n --reste)
  0 BEGIN >R 3DUP >R >R >R MU/MOD ROT R> R> ROT
  R> SWAP R> SWAP 0=
  WHILE 1+ >R -ROT 2DROP R>
  REPEAT
  DUP IF SWAP DUP CR 9 .R SWAP 15 .R
      ELSE DROP THEN
  >R 2SWAP 2DROP R> ;
: FACPR CR CR ." Diviseur     Coefficient" CR
  2 DIV 1+ DIV DROP 5 DIV
  BEGIN 2+ DIV 4 + DIV 3DUP DUP *D DU< UNTIL
  CR DROP 9 UD.R 1 15 .R ;
```

## LA PILE HEAP DE TURBO-PASCAL

par Bruno TREDEZ

Turbo-PASCAL

Pour tout langage compilé, les variables doivent être déclarées ainsi que leurs dimensions s'il s'agit de tableaux. Il en résulte que, même si une variable n'est utilisée que temporairement, elle occupe de la place en mémoire pendant toute l'utilisation du programme.

Mis à part l'assembleur, seuls les langages interpretés permettent la réutilisation d'une zone mémoire pour d'autres besoins (ex: instruction ERASE du Basic).

D'autre part, les variables "pointeur" ne peuvent généralement gérer que des adresses symboliques, ce qui ralentit l'exécution des programmes du fait de la conversion des adresses symboliques en adresses mémoire.

Turbo Pascal utilise deux piles :
1° La pile système
2° Une pile particulière dénommée HEAP destinée au stockage temporaire de variables dynamiques.

Celles-ci sont généralement du type "RECORD" et comportent plusieurs champs: clés, attributs ainsi qu'un ou plusieurs champs de type "Pointeur".

### LES TYPES RECORD ET POINTEUR

Ce sont deux types standard de variables. Exemple de déclaration :

```
 type
    pointemp =  ^employe; (*  variable pointeur pointant
sur employe *)
    employe = record;
      ident :integer;
      nom   :string[15];
      adr   :string[20];
      suiv  :pointemp;
    end; (employe)
```

Une variable pointeur est une variable d'adresse mémoire, elle occupe 4 octets (2 oct.:seg, 2 oct.:offs.) Les affectations entre pointeurs de même type sont licites.

Les deux piles utilisent (sauf instructions contraire),la totalité de la mémoire libre. La pile Heap est gérée au moyen de:

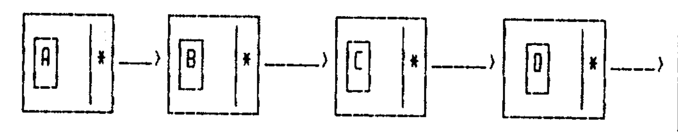
-Un pointeur système "heapptr" qui pointe en permanence sur le sommet de la pile.
-Trois procédures standard:
"new" alloue un nouvel emplacement au sommet de la pile (ex: new(employe) )
"mark (nom-var-pointeur)" affecte à une variable de type pointeur la valeur courante du pointeur de pile.
"release(nom de variable pointeur)" efface toutes les variables dynamiques à partir de cette adresse, libérant ainsi la place pour une nouvelle utilisation.

Au début d'un programme le pointeur heapptr est initialisé à l'adresse basse du segment pile et croit vers le haut à la rencontre de la pile CPU lors des réservations par NEW.
Les deux petits programmes POINT.PAS et STOCK.PAS sont des démonstrations de leur utilisation. Le premier effectue la saisie d'un fichier en mémoire centrale, empilant les enregistrements puis exécute un tri bulle sur les clés d'identification et liste le fichier dans l'ordre des clés. C'est le mécanisme de permutation qui est utilisé .

Le second gère un fichier disque physiquement trié. Il effectue le rappel dans la pile heap du fichier et permet la saisie de nouveaux enregistrements avec insertion immédiate par seule modification des pointeurs. Le fichier modifié est enfin réécrit sur le disque. Dans la pratique un tel mécanisme gèrerait non pas un fichier principal mais un fichier index associé. Les schémas suivants illustrent ces mécanismes.

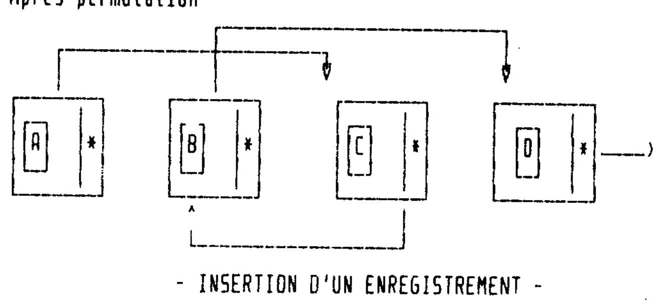
- PERMUTATION D'ENREGISTREMENTS -

Avant permutation




TEMP=POINT(A)  POINT(A)=POINT(B)  POINT(B)=POINT(C)
POINT(C)=TEMP

Après permutation




- INSERTION D'UN ENREGISTREMENT -

Avant insertion

A → B, C

POINT(A)=C   POINT(C)=B

Après insertion

A → C → B

LISTING DE POINT.PAS

```
Program POINTEURS; (* DEMONSTRATION DE TRI *)

type
  pointart = ^article;
  article  = record
    ref :integer;
    des :string[15];
    suiv:pointart;
    end;

var
  a:string[4];
  des:string[15];
  premier,nouveau,courant,c2,c3:pointart;
  r,s,n,i,j:integer;
  heaptop:^integer;
  inv:boolean;

label sortie;

procedure ajust; (* INVERSER SI SUIVANT < COURANT *)
begin
   c2:=courant^.suiv;c3:=c2^.suiv;
   r:=c2^.ref;s:=c3^.ref;
   if c2^.ref >c3^.ref then
   begin {3}
     courant^.suiv:=c3;nouveau:=courant;
     courant:=c2;
     courant^.suiv:=c3^.suiv;
     c3^.suiv:=c2;
     courant:=nouveau;inv:=true;
    end; {3}
   courant:=courant^.suiv;
   with courant^ do begin
   r:=ref;s:=suiv^.ref;
   end;
end; {ajust}

begin   {princip} (* SAISIE DES DONNEES *)
  writeln(' POUR SORTIR FRAPPER RETURN');writeln;
  premier:=nil;
  Mark(heaptop);
  n:=0;
  repeat
    write(' référence  ? ');readln (a);
    if a<>'' then
     begin
      val(a,r,s);
      n:=n+1;
      new (nouveau);
      nouveau^.ref:=r;
      write(' désignation ? ');readln(nouveau^.des);
      writeln;
      if premier=nil then
        begin
        premier:=nouveau;
        c2:=nouveau;
        end
      else
        c2^.suiv:=nouveau;
        nouveau^.suiv:=nil;
        c2:=nouveau;
    end;
  until a='';

(* TRI BULLE *)
for i:=n-1 downto 1 do
begin {1}
  inv:=false;
  with premier^ do
   if ref >suiv^.ref then
   begin {3}
    c2:=premier^.suiv;c3:=premier;
    premier^.suiv:=c2^.suiv;premier:=c2;
     premier^.suiv:=c3;
    inv:=true
   end; {3}
  courant:=premier;
  with courant^ do r:=suiv^.ref;
   for j:=2 to i do
   with courant^ do ajust;
   if inv=false then goto sortie;
 end; {1}

 sortie:

 writeln;
 while premier <> nil do
 with  premier ^ do
 begin
   writeln(ref:5,'  ',des);
   premier:=suiv;
 end;
release(heaptop);
end.
```

LISTING DE STOCK.PAS

```
program STOCK;  (* RAPPEL  INSERTION CLASSEE ET SAUVEGARDE
*)

type
  liste = record
    ref :integer;
    des :string[15];
  end;
  pointart = ^article;
  article  = record
    lar :liste;
    suiv:pointart;
  end;
type
  ficst =file of liste;
var
  stock : ficst;
  a:string[4];
  lst:liste;
  premier,courant,nouveau,temp:pointart;
  r,s:integer;
  baseheap:^integer;

(* SAISIE DES ENREGISTREMENTS *)
procedure saisie ;
begin {saisie}
  writeln(' POUR SORTIR FRAPPER RETURN');writeln;
  repeat
    write('reference  ? ');readln (a);
    if a<>'' then
     begin {1}
      val(a,r,s);
      new (nouveau);
      nouveau^.lar.ref:=r;
      write('désignation ? ');readln(nouveau^.lar.des);
      writeln;
      if premier=nil then
        begin
        premier:=nouveau;premier^.suiv:=nil;
        end
      else if r <premier^.lar.ref then
        begin
          nouveau^.suiv:=premier;
          premier:=nouveau;
          end
        else
          begin
          courant:=premier;
          repeat
           temp:=courant;
```

```
        courant:=courant^.suiv;
        until r<courant^.lar.ref;
        nouveau^.suiv:=courant;
        temp^.suiv:=nouveau;
       end;
   end; {1}
  until a='';
 end ; {saisie}
 (* SAUVEGARDE DU FICHIER CLASSE *)
 procedure save ;
 begin {save}
   rewrite (stock);
   writeln;
   while premier <> nil do
   with  premier ^ do
   begin
     write (stock,lar);
     premier:=suiv;
   end;
   close(stock);
 release(baseheap);
 end; {save}

 (* LECTURE ET EDITION DU FICHIER CLASSE *)
 procedure lecstock ;
 begin
   mark (baseheap);
   premier :=nil;
   writeln;
   reset (stock);
   while not eof(stock) do
   begin
     new (nouveau);
     read(stock,lst);
     nouveau^.lar :=lst;
     nouveau^.suiv :=nil;
     if premier = nil then premier := nouveau
     else temp^.suiv :=nouveau ;
     temp :=nouveau;
   end;
   close(stock);
 end;

 procedure attend;
   begin
   gotoxy(3,25);write('Frappez une touche');
   repeat until keypressed;
   clrscr;
   end;

procedure editst ;
begin
  clrscr;courant:=premier;
  while courant <> nil do
  begin
    writeln ( courant^.lar.ref:5,'  ',courant^.lar.des);
    if wherey=23 then attend;
    courant := courant^.suiv;
  end;
  attend;
end ; {editst}

procedure menu;
  begin
   clrscr;gotoxy(20,3);writeln('* MENU DES COMMANDES *');
   gotoxy(6,7);writeln('(1) Appel du fichier');
   gotoxy(6,9);writeln('(2) Edition du fichier');
   gotoxy(6,11);writeln('(3) Ajout dans le fichier');
   gotoxy(6,13);writeln('(4) Sauvegarde');
   gotoxy(6,15);writeln('(5) Retour au DOS');
  end;

var choix :char;
label deb , sortie;

begin {princip}
  assign(stock,'STK.FIC');
  textmode(3);textcolor(1);textbackground(7);
 deb:
  menu;
  repeat gotoxy(6,22);write(' Votre choix ? ');
  read(kbd,choix);write(choix);
  until choix in['1'..'5'];
    case choix of
      '1' :lecstock;
      '2' :editst;
      '3' :saisie;
      '4' :save;
      '5' :goto sortie;
    end;
    goto deb;
 sortie:
 end.
```